INSTRUCTIONS FOR USE OF THE ZXDB DISASSEMBLER/DEBUGGING PROGRAM

(C) COPYRIGHT 1981 BUG-BYTE

ZXDB occupies the first 4K of memory and is stored in a REM statement at line 1. To start ZXDB, just LOAD it (using the filename ZXDB) and RUN. This will cause a * prompt to appear, which tells you that ZXDB is waiting for a command letter to be typed.
A full list of commands, and their effects follows below.

NOTE: To use ZXDB and the ZXAS assembler together, use the following procedure (ignore this if you don't have ZXAS)
a) LOAD "ZXAS"
b) RUN
c) NEW
d) LOAD "ZXDB"
You will now have ZXDB in the bottom 4K of memory and ZXAS in the top 5K.
If you are not using ZXAS with ZXDB, then all lines after line 10 may be removed.

WHERE TO PUT YOUR MACHINE CODE

Within ZXDB, there is a free area of memory from 50C0 - 52FF (hex), which is the best place to store your code until it is bug-free. You can then transfer it to a separate REM statement so that ZXDB can be removed, giving more space for any accompanying BASIC program.

NOTE: All inputs to (and outputs from) ZXDB are made in hexadecimal, of which only the last four characters are valid. Pressing EDIT during any input will return the user to the * prompt.

COMMAND LIST

For each command, a letter is assigned, which must be typed in response to the * prompt in order to call the command. In the following instructions, you are then told which of the prompts = or > (or combination thereof) you will be given, and what ZXDB is expecting you to type, e.g.
= starting address
means that an = character appears on the screen, in answer to which you should type an address.
The best way to learn how to use ZXDB is to load it in and experiment with each command in turn. (They are given in order of complexity).

Q Leave ZXDB and return to BASIC.

G Execute a program.
= start of execution address.
JP 4100 at the end of a program will return to ZXDB, whereas JP 410 will return to BASIC.

V View memory in hexadecimal.
= starting address.
A line of 8 bytes of memory is displayed by ZXDB. Press newline to see next line. Press Q to return to the * prompt.

A Display memory as characters.
= start address.
A line of 32 bytes of memory is displayed as characters. Invalid codes are displayed as spaces. Press newline to see next line. Press Q to return to the = prompt for a new address, and Q again to return to the * prompt.

W Sets a display window.
= number of lines to be used.
This sets the number of lines used by ZXDB (counting from the bottom) for its display. Use this command when single stepping (see later) through a program which is using lines of the screen.

F Fill a block of memory.
> value to be placed in block.
= start of block.
= end of block.
Fills the defined block with the desired value.

E Hex loader.
= start address.
This is a simple hex loader in which the old value is shown and a new value is requested. Only inputs ending with newline will change the value.

C Compare two blocks of memory.
= start of first block.
= end of first block.
= start of second block.
ZXDB displays the differences between the two blocks of memory. At each pause, newline will continue the process. EDIT will return to the * prompt.

M Move a block of memory.
= start of memory block.
= end of block.
= start of destination block.
Moves a block of memory to a new location. If moving the block down memory, end the last input with I, otherwise end with newline.

D Disassemble a block of memory.
= start address.
Memory is disassembled by ZXDB, using standard Zilog mnemonics, with the following exceptions (which are mainly Intel 8080):-

ZILOG	THOSE USED
ld B,(IX+10)	LD B,IX+10
LD A,(1234)	LDA 1234
LD (1234),A	STA 1234
LD A,(DE)	LDAX DE
LD (DE),A	STAX DE
LD HL,(1234)	LHLD 1234
LD (1234),HL	SHLD 1234
LD SP,HL	SPHL
EX DE,HL	XCHG
EX AF,AF'	EX
JP (HL)	PCHL
EX (SP),HL	XTHL
IN A,(FE)	IN FE
OUT (FE),A	OUT FE
OUT (C),A	COUT A
IN A,(C)	CIN A

NOTE: HL is always termed M.

The above apply to other registers (where applicable), e.g. LD IX,(1234) is LIXD 1234 etc. etc.

Type a new address followed by newline to move the disassembler to that address. Type Q to return to the * prompt. Typing anything else will disassemble the next instruction.

S Search memory for an input string of characters.
After typing S, ZXDB waits for you to input the string that it is to search for, which is input as a string of hexadecimal values separated by full stops, and terminated by newline (the last character before newline must be a full stop). You then type the starting address for the search. ZXDB will then display the address of each occurrence of the string. at each pause, you may type Q to return to the * prompt (shift will stop a search that never finds its string).
Example: typing:-
2D.2A.31.31.34. (newline)
=4000 (newline)
will cause ZXDB to search RAM for the string "HELLO".
If four digit hex numbers are used, then the first two are taken as a mask for the second two. A match is defined as ((MEM) XOR (INPUT)) AND NOT MASK =0. In effect, if the mask bit is 1 then the corresponding input bit does not need to match memory.
Example: typing:-
CD.FF00.1F00. (newline)
=0000 (newline)
searches memory for any CALL to ROM.

Typing C at each pause allows you to change the string in memory by typing the new string in the same manner as the search string was input, and may be longer, shorter, or the same length as the original string.

B Set a breakpoint and execute a program.
= breakpoint address.
= address to start program execution.
This command is the most important of all ZXDB commands. A breakpoint is set in the users program, which when reached, causes the program to stop, displaying the contents of all registers, the states of the flags, the mnemonics of the instruction just executed, the next instruction, and a user-defined 8 byte memory window.
The program can now be continued in single step mode, in which one instruction at a time is executed and all registers re-displayed. This extremely powerful tool allows the user to see exactly what his program is doing (and if he sees it go wrong, to stop it before it crashes).
The display format is as follows:-

```
PC   SP   IX   IY   Flags set (carry, minus, zero, overflow)
(PC) (SP) (IX) (IY)  Mnemonic of instruction just executed
AF   BC   DE   HL
(AF) (BC) (DE) (HL)  Mnemonic of next instruction
WINDOW - 8 byte memory window
[(register pair) indicates the byte-reversed memory contents pointed to
by the register pair]
```

Here is a sample display:-

```
PC   SP   IX   IY
5100 7FEC 028F 4000   C  (the carry flag is set)
21DD 4100 29C3 C0FF   LD IX, 4523
AF   BC   DE   HL
5701 7323 56C0 5100
0000 0000 028F 21DD   LD IX, 4523
5300 76 00 0A 0E 00 F5 D4 1D
```

The above shows that e.g. H=51, L=00, (HL) is memory 5100 which holds DD, and memory 5101 holds 21 etc.

The following is a list of specific single-step commands available at each single-step (these are completely separate from the main ZXDB commands).
The command is entered as a single letter (terminating the parameter, if one is required - as shown by n before the command letter, e.g. enter 5100W to set the memory window to 5100)

newline executes the next single-step.

G lets the program run normally.

nW sets the memory window to n.

R used straight after a CALL executes the subroutine and regains single-step mode after the RETurn from the CALL.

nP sets the breakpoint to n.

Q returns to the * prompt.

L sets the breakpoint to the present (3 byte only) instruction and lets the program run (useful when in a loop).

O displays only the mnemonics of the instruction.

nN runs the next n instructions in single-step mode.

Z lets you set registers by entering the register followed by its required contents, e.g. AFF sets A to FF. Any of ABCDEFHL or M (M is (HL)) can be set. Type newline again to return to the single-step.

nT the program will only return to single step when a 16-bit register contains n or a memory reference is made to n.

GLADSTONE ELECTRONICS
301 FUHRMANN BLVD. BUFFALO, N.Y.
1736 AVENUE RD., TORONTO, ONT.